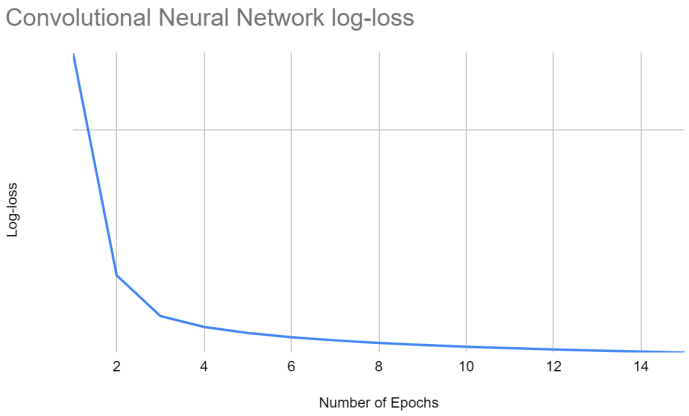
Exercise 1

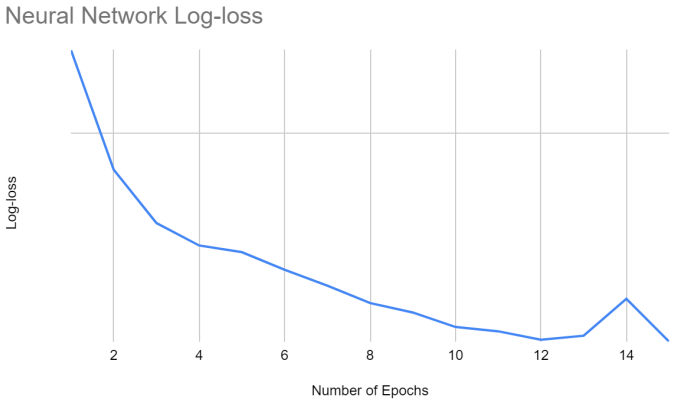
The NN model has two hidden layers with an input and an output layer. The input layer has 784 units because there are 784 pixels in each image. The loss was calculated as the cross entropy loss and was changed to log scale in evaluation for the visualization given below. The learning uses Stochastic Gradient Descent with a learning rate of 0.01. The hidden layers use ReLU as the activation function and softmax for the output function.

The CNN model has two hidden convolutional layers connected to a third hidden layer (to process the features given out from the convoluted layers) which is connected to an output layer to classify the images. The input layer has 1 unit because we are giving it a single image. The hyperparameters were the same as the NN model except for padding, kernel size and stride which was optimized to produce the best results.

Same data was fed to both of the models with the same training algorithm. The only difference was in training the Convolutional Neural Network, the images weren't flattened because it passes through the convoluted layers. The training uses forward pass and backward propagation to minimize the loss during the training process.
The precision and recall calculated is the weighted average because the model is a multi-class classifier. The models were trained for 15 epochs with 64 mini-batch size.

Model	Accuracy	Precision	Recall
NN	91.75%	91.73%	91.75%
CNN	97.71%	97.72%	97.71%





From the training log-loss data we can see above, as well as the evaluation metrics. We can see that the 2nd model (CNN) performs better than the feed-forward NN model. CNNs are designed for handling image classification tasks. The convolutional layers apply filters to the local regions of images so that it learns and detects patterns like edges and shapes. In a convolutional layer the same filter is applied on the entire image, so learned features can be detected anywhere in the image, sharing parameters to the whole network. This prevents overfitting and the model performs more efficiently. Because of these reasons I think that is why the CNN performed better than a feed-forward NN model for our MNIST dataset.

Exercise 2

The project is done using TensorFlow. For the project I have decided to train Neural Networks to predict stock prices. The data was acquired from yahoofinance package in python. The data included stocks data from 01/01/2010 to 12/01/2022. It consists of the following price values in USD: Open, High, Low, Close, Adj Close and Volume. The Volume column was removed from the dataset as it doesn't represent any monetary features and seemed irrelevant when trying to predict stock prices. The data was preprocessed by normalizing the dataset from a range of 0 to 1 and sequentially storing the data for each stock so that it contained data from the past 60 days. 80-20 train-test split was done and the model was evaluated with test data ranging from 04/2020 to 12/2022

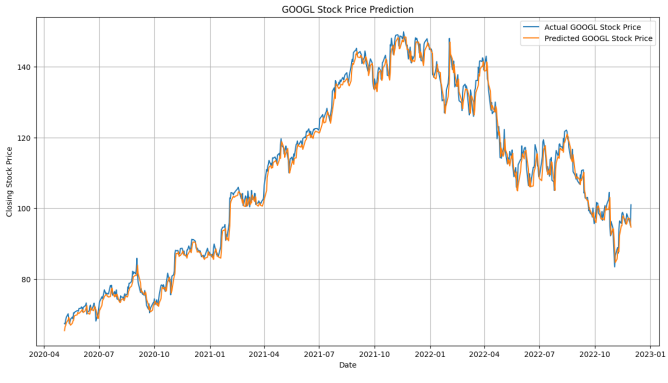
The first model I trained is an LSTM model which inputs the past 60 days of stocks data and predicts the stock price of the next day. It uses two LSTM layers of 100 units connected to a Dense layer with 25 units and ReLU activation. It is then fully connected to a Dense Output layer with softmax activation. The model uses 'Adam' as the optimizer and the loss function is the Mean Squared Error. The model is trained for 15 epochs with a batch size of 32. The training is sequentially done for each stock data as well as the evaluation. For example, the model is trained for each stock for all of the sequential data.

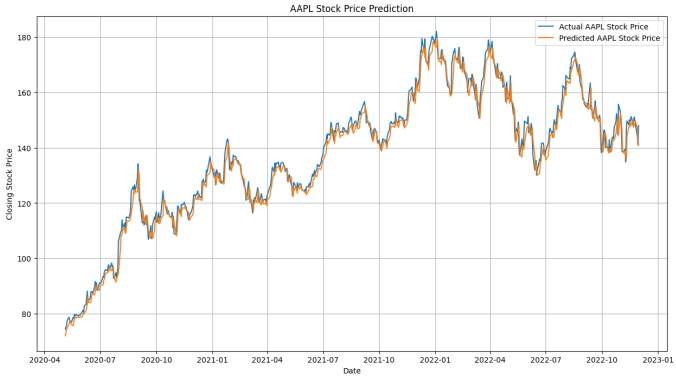
The second model I trained is a Recurrent Neural Network with Gated Recurrent Units (GRUs). It consists of an input GRU layer with 120 units connected to another GRU with 60 units. The model is then connected to 2 dense layers with 50 and 25 units respectively. Finally it is connected to a dense output layer with softmax activation. Same optimizer, loss function, batch size and epochs are used during the training process as with the model before.

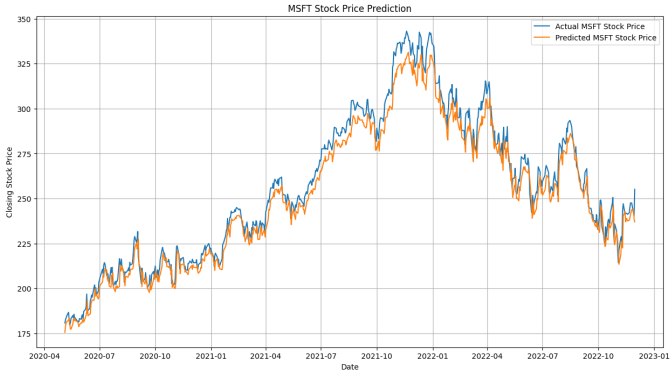
Evaluation

Since this is a regression problem, accuracy, precision and recall is irrelevant hence I used Root Mean Squared error as the evaluation metric for the two models. Each model was tested to predict closing stock price data for each stock. The actual and (denormalized) predicted values were graphed and compared for the two models below. The LSTM model had a RMSE = 5.90155 whereas the RNN model had a RMSE = 4.54563. The RNN model trained much faster (almost in half the time) and better than the LSTM model. LSTMs generally perform well with long-range dependencies. The lookback was set to 60 days for this model and is relatively low. The LSTM model had a very low training loss than the test loss which means that the model was overfitting on the training data. The RNN model had an extra hidden Dense layer and had more units in the Gated Recurrent Layers. I think that the reason why the GRU-RNN model performed better is because of the low lookback value and did not overfit on the training data. I believe that with a higher lookback value, the LSTM would perform better than the GRU-RNN model.

In conclusion, stock price prediction is an inherently difficult task, and the performance of any model is likely to be influenced by factors such as market volatility, news events, and investor sentiment, which may not be easily captured by historical price data alone. Adding more features like technical indicators e.g moving average, RSI, and MACD would allow for a more accurate model. Additionally, having a LLM analyze stock market sentiments and feeding the output to the model would lead to a much better model that can predict future stock prices.







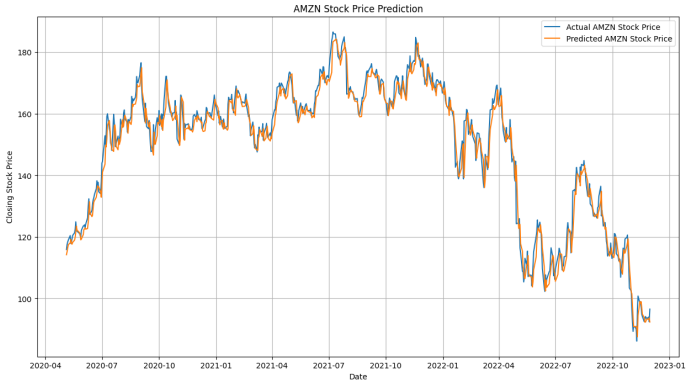
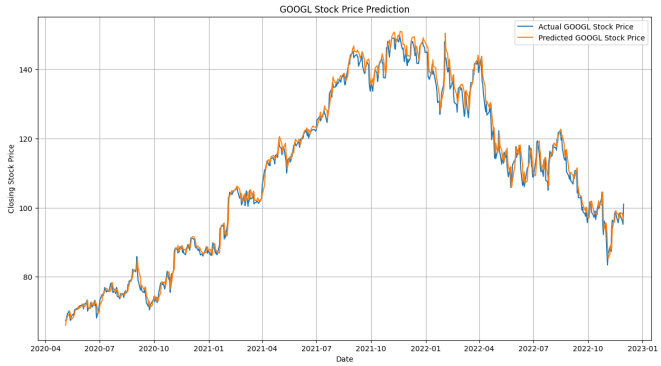
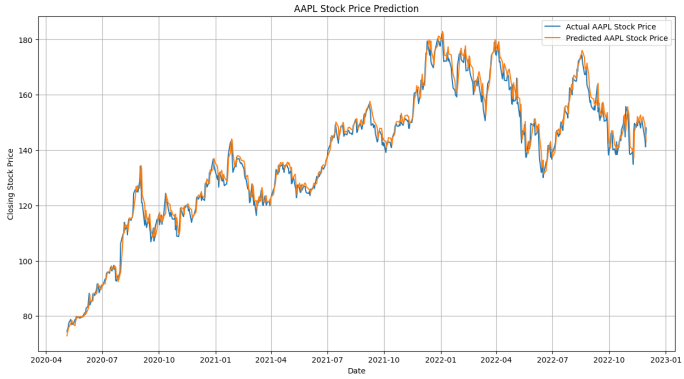
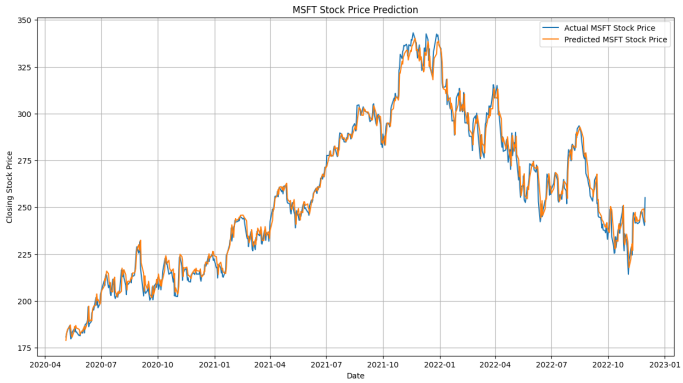


Fig: LSTM model predictions







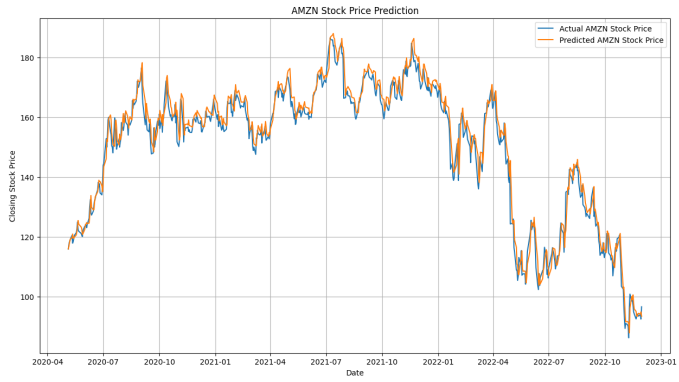


Fig: GRU-RNN predictions